Python_Packages

November 15, 2019

1 Numpy : creating and manipulating numerical data

NumPy is a library of Pythons that helps to easily handle matrixes or generally large-scale multidimensional arrays.NumPy provides efficient functions for numerical calculations in addition to data structures.

Crate array

```
In [1]: import numpy as np
        a = np.array([0, 1, 2, 3])
        print("a :",a)
```

```
a : [0 1 2 3]
```

2 Memory-efficient container that provides fast numerical operations.

It is very efficient to overtake Numpy's range rather than to calculate it through the built-in function, range.

```
In [2]: L = range(1000)
        %timeit [i**2 for i in L]
```

```
271 µs ś 8.84 µs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

```
In [3]: a = np.arange(1000)
        %timeit a**2
```

```
1.26 µs ś 1.85 ns per loop (mean ś std. dev. of 7 runs, 1000000 loops each)
```

3 Creating arrays

1-D : one-dimensional Array

```
In [4]: a = np.array([0, 1, 2, 3])
        print("a :",a)
        print("a's ndim :", a.ndim)
        print("a's size:", a.size)
        print("a's shape:",a.shape)
        print("a's len:", len(a))
```

```
a : [0 1 2 3]
a's ndim : 1
a's size: 4
a's shape: (4,)
a's len: 4
```

2-D, 3-D, ... : two-dimensional, three-dimensional Array

```
In [5]: b = np.array([[0, 1, 2], [3, 4, 5]]) # 2 x 3 array
        print("b :\n",b)
        print("b's ndim :", b.ndim)
        print("b's size:", b.size)
        print("b's shape:",b.shape)
        print("b's len:", len(b))
```

```
b :
 [[0 1 2]
 [3 4 5]]
b's ndim : 2
b's size: 6
b's shape: (2, 3)
b's len: 2
```

```
In [6]: c = np.array([[[1], [2]], [[3], [4]]])
        print("c :\n",c)
        print("c's ndim :", c.ndim)
        print("c's size:", c.size)
        print("c's shape:",c.shape)
        print("c's len:", len(c))
```

```
c :
 [[[1]
  [2]]

 [[3]
  [4]]]
c's ndim : 3
c's size: 4
c's shape: (2, 2, 1)
c's len: 2
```

4 Evenly spaced

When creating regular arrangements, the np.arrange (start, end, and interval) function is used; if only one value is given, the starting value is set to 0 and the interval to 1.

```
In [7]: a = np.arange(10)
        a

Out[7]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [8]: b = np.arange(1, 9, 2)
        b

Out[8]: array([1, 3, 5, 7])
```

5 By number of points

When creating an array between intervals, make it np.linspace (start, end, number); using the endpoint option, you can determine whether to include the endpoint.

```
In [9]: c = np.linspace(0, 1, 6) # start, end, num-points
        c

Out[9]: array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])

In [10]: d = np.linspace(0, 1, 5, endpoint=False)
         d

Out[10]: array([0. , 0.2, 0.4, 0.6, 0.8])
```

6 Common arrays

You can create a variety of matrices, such as an array of elements with a value of one element, an array of elements with a value of zero, and a unit matrix.

```
In [11]: a = np.ones((2, 2)) # reminder: (2, 2) is a tuple
         a

Out[11]: array([[1., 1.],
                [1., 1.]])

In [12]: b = np.zeros((2, 2))
         b

Out[12]: array([[0., 0.],
                [0., 0.]])

In [13]: c = np.eye(2)
         c
```

```
Out[13]: array([[1., 0.],
                [0., 1.]])

In [14]: d = np.diag(np.array([1, 2, 3, 4]))
         d

Out[14]: array([[1, 0, 0, 0],
                [0, 2, 0, 0],
                [0, 0, 3, 0],
                [0, 0, 0, 4]])
```

7 Random numbers

uniform in [0, 1] : Random number generation in uniform distribution

```
In [15]: a = np.random.rand(4)
         a

Out[15]: array([0.49050823, 0.90898709, 0.17690019, 0.31417126])
```

Gaussian : Random number generation in Gaussian distribution

```
In [16]: b = np.random.randn(4)
         b

Out[16]: array([-1.2280953 ,  0.42113584, -0.08267679, -1.93946032])
```

Setting the random seed : The seed is given and the random number production can be clinched.

```
In [17]: np.random.seed(1234)
```

8 Indexing and slicing

```
In [18]: a = np.arange(10)
         a

Out[18]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Extract only values with 0,1,2 indexes in a

```
In [19]: a[0], a[2], a[-1]

Out[19]: (0, 2, 9)
```

Step forward one space from the back

```
In [20]: a[::-1]

Out[20]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

4x4 diag array generation

```
In [21]: a = np.diag(np.arange(4))
         a

Out[21]: array([[0, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 2, 0],
                [0, 0, 0, 3]])
```

return of (1,1) of a

```
In [22]: a[1, 1]

Out[22]: 1
```

transform the value of (1,1) of a to 10

```
In [23]: a[1, 1] = 10
         a

Out[23]: array([[ 0,  0,  0,  0],
                [ 0, 10,  0,  0],
                [ 0,  0,  2,  0],
                [ 0,  0,  0,  3]])
```

one line output of a

```
In [24]: a[1]

Out[24]: array([ 0, 10,  0,  0])
```

1 row output of a (the Pythonian code)

```
In [25]: a[1,:]

Out[25]: array([ 0, 10,  0,  0])
```

9 Copies and views

```
In [26]: a = np.arange(10)
         a

Out[26]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

B shares a array by skipping two spaces

```
In [27]: b = a[::2]
         b

Out[27]: array([0, 2, 4, 6, 8])
```

Whether a and b share memory

```
In [28]: np.may_share_memory(a, b)
```

```
Out[28]: True
```

Change the 0th value of b to 12

```
In [29]: b[0] = 12
```

```
In [30]: b
```

```
Out[30]: array([12,  2,  4,  6,  8])
```

Since a and b share memory, change the 0th of a to 12

```
In [31]: a
```

```
Out[31]: array([12,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

```
In [32]: a = np.arange(10)
         a
```

```
Out[32]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

C arrays are copied by skipping two sets of a array

```
In [33]: c = a[::2].copy()
         c
```

```
Out[33]: array([0, 2, 4, 6, 8])
```

Change the 0th value of c to 12

```
In [34]: c[0] = 12
         c
```

```
Out[34]: array([12,  2,  4,  6,  8])
```

C is stored in a new memory because it is a copy of a. Therefore, the 0th value of a is not changed.

```
In [35]: a
```

```
Out[35]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

memory sharing whether or not a and c are shared

```
In [36]: np.may_share_memory(a, c)
```

```
Out[36]: False
```

10 Fancy indexing

7 random number generation (integer) between 0 ~ 20

```
In [37]: a = np.random.random_integers(0, 20, 7)
         a
```

```
C:\Users\user\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: This fu
  """Entry point for launching an IPython kernel.
```

```
Out[37]: array([15, 19,  6, 12, 20, 15, 17])
```

determining whether each element is a multiple of 3

```
In [38]: (a % 3 == 0)
```

```
Out[38]: array([ True, False,  True,  True, False,  True, False])
```

In the mask, each element is determined to be a multiple of 3 to store True, False, and in the extract_from_a, only elements with a mast are stored

```
In [39]: mask = (a % 3 == 0)
         extract_from_a = a[mask] # or, a[a%3==0]
         extract_from_a
```

```
Out[39]: array([15,  6, 12, 15])
```

```
In [40]: a = np.arange(0, 10, 1)
         a
```

```
Out[40]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Returns the second, third, second, fourth, second value of a

```
In [41]: a[[2, 3, 2, 4, 2]]
```

```
Out[41]: array([2, 3, 2, 4, 2])
```

transform the ninth and seventh of a to -1

```
In [42]: a[[9, 7]] = -1
         a
```

```
Out[42]: array([ 0,  1,  2,  3,  4,  5,  6, -1,  8, -1])
```

11 Numerical operations on arrays

Add 1 to all element values of a

```
In [43]: a = np.array([1, 2, 3, 4])
         a + 1
```

```
Out[43]: array([2, 3, 4, 5])
```

Multiplies all element values of a by 2

```
In [44]: 2**a
```

```
Out[44]: array([ 2,  4,  8, 16], dtype=int32)
```

b = [2,2,2,2] and calculation of each of the same indexing elements

```
In [45]: b = np.ones(4) + 1
         a - b
```

```
Out[45]: array([-1.,  0.,  1.,  2.])
```

product by indexing elements of the same

```
In [46]: a * b
```

```
Out[46]: array([2., 4., 6., 8.])
```

$j = [0,1,2,3,4]$

Add 1 to each in j, calculate it as a square of 2 and then .. $j = [0,1,2,3,4]$

$j = [2^{(0+1)} - 0,\ 2^{(1+1)} - 1,\ 2^{(2+1)} - 2,\ 2^{(3+1)} - 3,\ 2^{(4+1)} - 4] = [2,\ 3,\ 6,\ 13,\ 28]$

```
In [47]: j = np.arange(5)
         2**(j + 1) - j
```

```
Out[47]: array([ 2,  3,  6, 13, 28])
```

product of each element $c = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

$c * c = \begin{pmatrix} 1 \times 1 & 1 \times 1 \\ 1 \times 1 & 1 \times 1 \end{pmatrix}$

```
In [48]: c = np.ones((2, 2))
         c * c
```

```
Out[48]: array([[1., 1.],
                [1., 1.]])
```

matrix product $c = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

$c \cdot c = \begin{pmatrix} 1 \times 1 + 1 \times 1 & 1 \times 1 + 1 \times 1 \\ 1 \times 1 + 1 \times 1 & 1 \times 1 + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$

```
In [49]: c.dot(c)
```

```
Out[49]: array([[2., 2.],
                [2., 2.]])
```

Determining whether each element of a and b coincides with each other

```
In [50]: a = np.array([1, 2, 3, 4])
         b = np.array([4, 2, 2, 4])
         a == b

Out[50]: array([False,  True, False,  True])
```

the discrimination each elements of a and b are a big than b

```
In [51]: a > b

Out[51]: array([False, False,  True, False])
```

Determining whether a and b are all the same

```
In [52]: a = np.array([1, 2, 3, 4])
         b = np.array([4, 2, 2, 4])
         c = np.array([1, 2, 3, 4])
         np.array_equal(a, b)

Out[52]: False
```

Determines whether a and c are all the same

```
In [53]: np.array_equal(a, c)

Out[53]: True
```

The element of a and b is the logical operator "or" cognitive discriminant (True even if only one of them is one, and False if both are zero)

```
In [54]: a = np.array([1, 1, 0, 0], dtype=bool)
         b = np.array([1, 0, 1, 0], dtype=bool)
         np.logical_or(a, b)

Out[54]: array([ True,  True,  True, False])
```

The elements of a and b are the logical operator "and" cognitive discriminant (both only 1 to True, the rest being False)

```
In [55]: np.logical_and(a, b)

Out[55]: array([ True, False, False, False])
```

12 Reduction

Calculation of the sum of elements of x

```
In [56]: x = np.array([1, 2, 3, 4])
         np.sum(x)

Out[56]: 10

In [57]: x = np.array([[1, 1], [2, 2]])
         x

Out[57]: array([[1, 1],
                [2, 2]])
```

Calculation of sums based on columns

```
In [58]: x.sum(axis = 0)

Out[58]: array([3, 3])
```

Calculation of the sum of the 0th column and the 1st column, respectively

```
In [59]: x[:, 0].sum(), x[:, 1].sum()

Out[59]: (3, 3)
```

Calculation of sums based on rows

```
In [60]: x.sum(axis=1)

Out[60]: array([2, 4])
```

Calculation of the sum of the 0th row and the 1st row respectively

```
In [61]: x[0, :].sum(), x[1, :].sum()

Out[61]: (2, 4)
```

3D Array random number generation of 2x2x2

```
In [62]: x = np.random.rand(2, 2, 2)
         x

Out[62]: array([[[0.27259261, 0.27646426],
                 [0.80187218, 0.95813935]],

                [[0.87593263, 0.35781727],
                 [0.50099513, 0.68346294]]])
```

x.sum(axis=2)[0, 1] : x[0,1,0] + x[0,1,1]

```
In [63]: x.sum(axis=2)[0, 1]

Out[63]: 1.7600115312187246

In [64]: x[0, 1, :].sum()

Out[64]: 1.7600115312187246

In [65]: x = np.array([1, 2, 3, 1])
         y = np.array([[1, 2, 3], [5, 6, 1]])

In [66]: x

Out[66]: array([1, 2, 3, 1])

In [67]: y

Out[67]: array([[1, 2, 3],
                [5, 6, 1]])
```

calculation of the mean of x

```
In [68]: x.mean()

Out[68]: 1.75
```

calculation of the center value of x

```
In [69]: np.median(x)

Out[69]: 1.5
```

Calculation of the median of x based on the last axis

```
In [70]: np.median(y, axis=-1) # last axis

Out[70]: array([2., 5.])
```

calculation of the standard deviation of x

```
In [71]: x.std() ## full population standard dev.

Out[71]: 0.82915619758885
```

13 Pandas data-frame

pandas is a data frame library

Csv file import, sep= option specifies delimiter, na_values= option specifies how to display missing value

```
In [72]: import pandas
         data = pandas.read_csv('brain_size.csv', sep=';',na_values=".")
         data.head()
```

```
Out[72]:    Unnamed: 0  Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
         0           1  Female   133  132  124   118.0    64.5     816932
         1           2    Male   140  150  124     NaN    72.5    1001121
         2           3    Male   139  123  150   143.0    73.3    1038437
         3           4    Male   133  129  128   172.0    68.8     965353
         4           5  Female   137  132  134   147.0    65.0     951545
```

14 Creating from arrays

After data generation through array, it is converted to pandas data frame.

```
In [73]: import pandas as pd
         import numpy as np
         t = np.linspace(-6, 6, 20) # -6 6   20
         sin_t = np.sin(t)
         cos_t = np.cos(t)
         data1 = pd.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t})
         data1.head()
```

```
Out[73]:           t       sin       cos
         0 -6.000000  0.279415  0.960170
         1 -5.368421  0.792419  0.609977
         2 -4.736842  0.999701  0.024451
         3 -4.105263  0.821291 -0.570509
         4 -3.473684  0.326021 -0.945363
```

15 Manipulating data

data frame preview

```
In [74]: data = pandas.read_csv('brain_size.csv', sep=';',na_values=".")
         data.head()
```

```
Out[74]:    Unnamed: 0  Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
         0           1  Female   133  132  124   118.0    64.5     816932
         1           2    Male   140  150  124     NaN    72.5    1001121
         2           3    Male   139  123  150   143.0    73.3    1038437
         3           4    Male   133  129  128   172.0    68.8     965353
         4           5  Female   137  132  134   147.0    65.0     951545
```

data frame structure confirmation

```
In [75]: data.shape
```

```
Out[75]: (40, 8)
```

Dataframe column (variable name) confirmation

```
In [76]: data.columns
```

```
Out[76]: Index(['Unnamed: 0', 'Gender', 'FSIQ', 'VIQ', 'PIQ', 'Weight', 'Height',
                'MRI_Count'],
               dtype='object')
```

Output only specific indexing of a specific column

```
In [77]: print(data['Gender'][0:5])
```

```
0    Female
1      Male
2      Male
3      Male
4    Female
Name: Gender, dtype: object
```

Calculate the mean value by giving the condition in a specific column

```
In [78]: data[data['Gender'] == 'Female']['VIQ'].mean()
```

```
Out[78]: 109.45
```

Output of Summary Statistics for each column in Data Frame

```
In [79]: data.describe()
```

```
Out[79]:        Unnamed: 0        FSIQ         VIQ        PIQ      Weight     Height  \
         count   40.000000   40.000000   40.000000   40.00000   38.000000  39.000000
         mean    20.500000  113.450000  112.350000  111.02500  151.052632  68.525641
         std     11.690452   24.082071   23.616107   22.47105   23.478509   3.994649
         min      1.000000   77.000000   71.000000   72.00000  106.000000  62.000000
         25%     10.750000   89.750000   90.000000   88.25000  135.250000  66.000000
         50%     20.500000  116.500000  113.000000  115.00000  146.500000  68.000000
         75%     30.250000  135.500000  129.750000  128.00000  172.000000  70.500000
         max     40.000000  144.000000  150.000000  150.00000  192.000000  77.000000

                    MRI_Count
         count   4.000000e+01
         mean    9.087550e+05
         std     7.228205e+04
```

```
min    7.906190e+05
25%    8.559185e+05
50%    9.053990e+05
75%    9.500780e+05
max    1.079549e+06
```

16 groupby

Binding data based on a specific column

```
In [80]: groupby_gender = data.groupby('Gender')
```

Calculation of Mean Values of Each Variable by Gender

```
In [81]: groupby_gender.mean()
```

```
Out[81]:         Unnamed: 0   FSIQ     VIQ     PIQ      Weight     Height  MRI_Count
         Gender
         Female       19.65  111.9  109.45  110.45  137.200000  65.765000   862654.6
         Male         21.35  115.0  115.25  111.60  166.444444  71.431579   954855.4
```

Calculation of Mean Values of VIQ Variables by Gender

```
In [82]: for gender, value in groupby_gender['VIQ']:
             print((gender, value.mean()))
```

```
('Female', 109.45)
('Male', 115.25)
```

```
In [83]: groupby_gender.boxplot(column=['FSIQ', 'VIQ', 'PIQ'])
```

```
Out[83]: Female         AxesSubplot(0.1,0.15;0.363636x0.75)
         Male      AxesSubplot(0.536364,0.15;0.363636x0.75)
         dtype: object
```

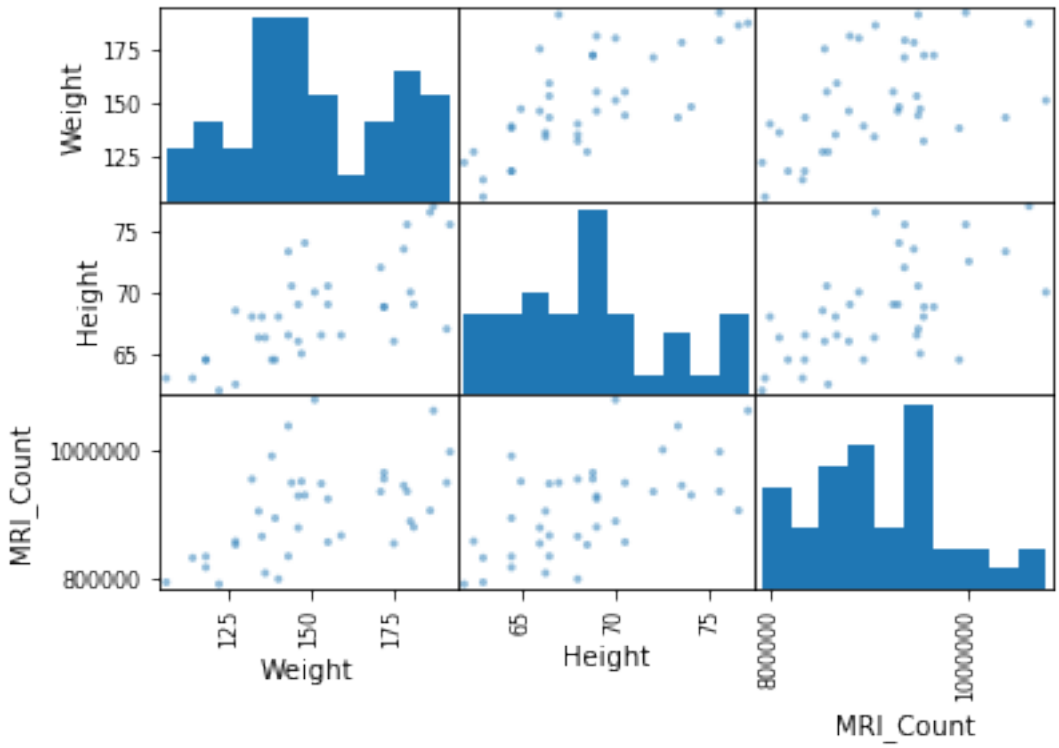
17 Plotting data

```
In [84]: from pandas.tools import plotting
         plotting.scatter_matrix(data[['Weight','Height','MRI_Count']])
```

```
C:\Users\user\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: 'pandas.tools
```

```
Out[84]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFCBE550>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFCE5B70>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFD0FEF0>],
```

```
 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFD3F400>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFD67978>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFD8CEF0>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFDBF4A8>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFDE6A58>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000029FBFDE6A90>]],
dtype=object)
```



18 Scipy

SciPy is a free-open source Python library used in scientific computing and technical computing

Create a matrix with all elements of the 3x3 matrix

```
In [85]: from scipy import io as spio
         a = np.ones((3, 3))
         a

Out[85]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

Save the created a matrix as a file

```
In [86]: spio.savemat('file.mat',{'a': a}) #save mat expects a dic
```

Bring in a stored matrix

```
In [87]: data = spio.loadmat('file.mat', struct_as_record=True)
         data['a']

Out[87]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])

In [88]: from scipy import linalg
         arr = np.array([[1, 2], [3, 4]])
         arr

Out[88]: array([[1, 2],
                [3, 4]])
```

(Matrix Determinant): np.linalg.det(x)

Use the indicator determinant (determinant, abbreviated det) as a way to determine whether the inverse exists.If this matrix is not '0', there is a retromatrix, and if this matrix is '0', there is no retromatrix.

The arr's Matrix Determinant is -2, therefore it has inverse matrix.

```
In [89]: linalg.det(arr)

Out[89]: -2.0

In [90]: arr = np.array([[3, 2], [6, 4]])
```

The above arr does not have a inverse matrix (6.661338147750939e-16 = 0).

```
In [91]: linalg.det(arr)

Out[91]: 6.661338147750939e-16
```

inverse matrix calculation

```
In [92]: arr = np.array([[1, 2], [3, 4]])
         iarr = linalg.inv(arr)
         iarr

Out[92]: array([[-2. ,  1. ],
                [ 1.5, -0.5]])
```

Determining whether two matrices are the same matrices: np.allclose()

```
In [93]: arr

Out[93]: array([[1, 2],
                [3, 4]])

In [94]: iarr

Out[94]: array([[-2. ,  1. ],
                [ 1.5, -0.5]])

In [95]: np.dot(arr, iarr)

Out[95]: array([[1.0000000e+00, 0.0000000e+00],
                [8.8817842e-16, 1.0000000e+00]])

In [96]: np.eye(2)

Out[96]: array([[1., 0.],
                [0., 1.]])

In [97]: np.allclose(np.dot(arr, iarr), np.eye(2))

Out[97]: True
```

19 Hypothesis Testing: comparing two groups

```
In [98]: from scipy import stats
         import pandas
         data = pandas.read_csv('brain_size.csv', sep=';',na_values=".")
         data.head()
```

Out[98]:

	Unnamed: 0	Gender	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
0	1	Female	133	132	124	118.0	64.5	816932
1	2	Male	140	150	124	NaN	72.5	1001121
2	3	Male	139	123	150	143.0	73.3	1038437
3	4	Male	133	129	128	172.0	68.8	965353
4	5	Female	137	132	134	147.0	65.0	951545

19.0.1 1-sample t-test: testing the value of a population mean

```
In [99]: stats.ttest_1samp(data['VIQ'], 0)

Out[99]: Ttest_1sampResult(statistic=30.088099970849328, pvalue=1.3289196468728067e-28)
```

19.0.2 2-sample t-test: testing for difference across populations

```
In [100]: female_viq = data[data['Gender'] == 'Female']['VIQ']
          male_viq = data[data['Gender'] == 'Male']['VIQ']
          stats.ttest_ind(female_viq, male_viq)

Out[100]: Ttest_indResult(statistic=-0.7726161723275011, pvalue=0.44452876778583217)
```

19.0.3 Paired t-test: repeated measurements on the same individuals

```
In [101]: stats.ttest_rel(data['FSIQ'], data['PIQ'])
```

```
Out[101]: Ttest_relResult(statistic=1.7842019405859857, pvalue=0.08217263818364236)
```

20 Linear models

```
In [102]: import numpy as np
          x = np.linspace(-5, 5, 20)
          np.random.seed(1)
```

20.0.1 normal distributed noise

```
In [103]: y = -5 + 3*x + 4 * np.random.normal(size=x.shape)
```

20.0.2 Create a data frame containing all relevant variables

```
In [104]: data = pandas.DataFrame({'x': x, 'y': y})
```

20.0.3 OLS fit

```
In [105]: from statsmodels.formula.api import ols
          model = ols("y ~ x", data).fit()
          print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.804
Model:                            OLS   Adj. R-squared:                  0.794
Method:                 Least Squares   F-statistic:                     74.03
Date:                Fri, 15 Nov 2019   Prob (F-statistic):           8.56e-08
Time:                        18:31:32   Log-Likelihood:                -57.988
No. Observations:                  20   AIC:                             120.0
Df Residuals:                      18   BIC:                             122.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -5.5335      1.036     -5.342      0.000      -7.710      -3.357
x              2.9369      0.341      8.604      0.000       2.220       3.654
==============================================================================
Omnibus:                        0.100   Durbin-Watson:                   2.956
Prob(Omnibus):                  0.951   Jarque-Bera (JB):                0.322
Skew:                          -0.058   Prob(JB):                        0.851
Kurtosis:                       2.390   Cond. No.                         3.03
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```